Enabling Rapid and Reproducible Modeling via Software Development Best Practices

D.W. Munster[1] and S.A. Bostic[2]
[1]NASA Glenn Research Center, Cleveland, OH
[2]BQMI, Cleveland, OH

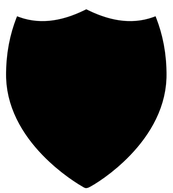
I. Introduction

As the availability of compute resources increases and research supporting human space travel requires the intersection of increasingly complex fields, computational modeling becomes an attractive method for probing otherwise inscrutable topics. However, these models are often developed on an ad-hoc basis by researchers or teams whose expertise lies in the subject-matter area and may have limited experience in software development. By recognizing that scientific software is software, we may leverage best practices in software development to build computational models that are:

- 

Reproducible

The ability to reproduce an experiment is a cornerstone of the scientific process. For computational modeling, this can require keeping track of parameters used to initialize random number generation, the exact values of parameters and their provenance, the version of the code utilized, and tracking or bundling dependencies so that others may set up an identical environement.

- 

Robust

An implementation that is developed defensively and with sufficient testing increases confidence in the model results and will help catch mistakes and misconfigurations before those can corrupt the results of the study.

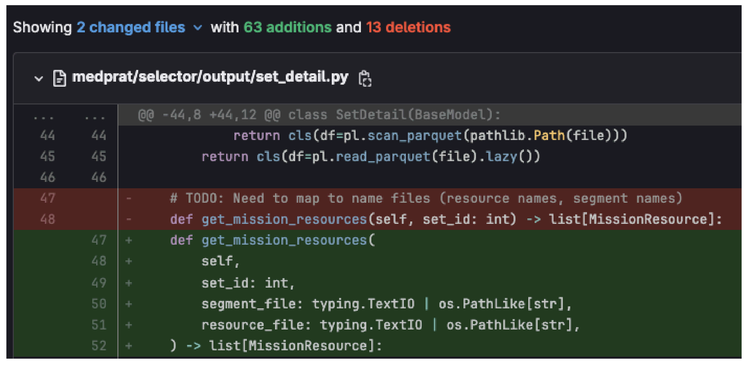
- 

Rapidly Developed

High quality codebases with robust testing encourage rapid development by enabling developers to proceed "fearlessly", leveraging the tooling that has been built and knowing that testing will help discover mistakes.

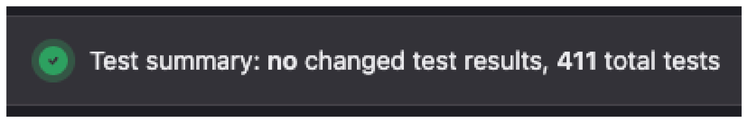
II. Best Practices

Version Control



Tracking the changes to the code over time is an easily adopted practice that aids reproducibility. Distributed version control systems, such as Git, can be utilized on an individual researcher's machine without any centralized or external resources. While centralized platforms can aid collaboration, changes to the code can be tracked locally which enables exact reproduction of the code at a given time. Many tools, such as MATLAB and VSCode, have built-in support for Git or other version control systems.

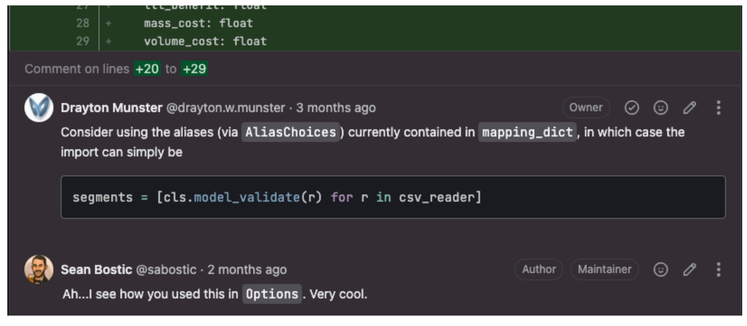
Testing



Devloping a test suite is key for building robust and rapidly developed software. Having tests to assert the proper functioning of the code enables improvements to be written with confidence that they haven't compromised correctness. While the following types of tests are not exhaustive, they represent a good starting point:

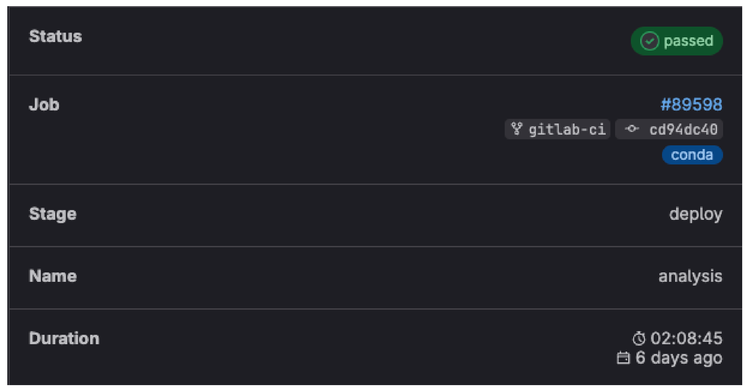
- Unit Tests: The testing of small, independent units of code, often at the individual function level. Both correct and incorrect input should be tested to ensure the system behaves as expected.
- Regression Tests: Tests that assert that a particular behavior/outcome has not changed. These can encompass a wide variety of goals, such as
  - Systems/Integration behavior (e.g., a simple case of the model to ensure it matches an expected outcome)
  - Performance (e.g., ensuring a change hasn't significantly increased runtime)
  - Bugfixes (e.g., ensuring a bug stays fixed)

Code Review



When working with multiple collaborators, requiring one or more people to review and approve suggested changes helps to build robust code. A set of "fresh eyes" may be able to identify edge cases that were missed, sections that need additional explanation, or offer improvements. Code review also increases awareness of the code base which can identify common functionality and other refactoring opportunities.

Automation



Automation is an effective way to generate reproducible analyses since this requires exactly specifying each step of the process for the computer to execute. When the input and output data are tied to an automated execution, it is trivial to re-execute the steps in the automation to reproduce the result.

III. Resources

Better Scientific Software (https://bssw.io/):

Git (https://git-scm.com/):

drayton.w.munster@nasa.gov, sean.a.bostic@nasa.gov